

**工程技术文档**

**版本(V1.0.0)**

睿智融科XtTraderApi使用手册

Last updated: 2021-05-07

修 订 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 柜台启用版本 | 文档状态 | 日 期 | 说 明 |
| 1.0前 |  |  |  |  |
| 1.0 |  | A | 20141211 | 增加此接口说明 |
| 1.1 |  | A | 20160901 | 增加详细接口说明 |
| 1.2 |  | A | 20160923 | 增加信用、期权说明 |
| 1.3 |  | M | 20161223 | 修改部分文档说明 |
| 1.3.1 |  | M | 20210507 | 修改部分文档说明 |

状态标识：C–Created A–Added M–Modified D–Deleted

**简介**

量化交易作为中国市场的新型投资方法，越来越受到投资者的亲睐。原有的交易系统并没有针对量化交易进行设计和优化，XtTraderApi应运而生。顾名思义，XtTraderApi为投顾提供了用于量化和主观交易的可编程接口。在交易程序中调用这些接口，投资者可实现用户登录/登出、查询、下单、撤单等操作。相对于迅投交易系统的原先的C/S模型下，XtTraderApi类似于一个瘦客户端，使得投资者可以平滑地访问迅投服务。

**所有权声明**

本文档的内容将做定期性的变动，且不另行通知。更改的内容将会补充到本手册中。

本文档的版权属北京睿智融科控股有限公司所有，受中华人民共和国法律的保护。本文档所含的任何构思、设计、及其他技术信息均属本公司所有，受中华人民共和国法律的保护。未经本公司书面同意，任何单位和个人不得擅自摘抄、全部或部分复制本书内容，或者以其他任何方式使第三方知悉。

1、调用或参看通过非正式渠道获取的睿智融科XtTraderApi接口及相关文档所造成的一切损失，睿智融科概不负责，亦不承担任何法律责任。

2、睿智融科力求保证每个业务接口的正确性，对第三方系统如何调用不做担保。

3、专用接口是为专用客户定做的接口，不适用于其他客户，若其他客户因调用造成一切损失，睿智融科概不负责，亦不承担任何法律责任。

**支持和服务**

如果您在使用本公司XtTraderApi接口时遇到困难，请先仔细阅读本手册，其中对各接口提供了详尽的描述。

我司将努力改进接口和相关的说明册，使其更便于使用，您的意见对于我们的工作非常重要，您可以用您喜欢的任何方式告诉我们。我们技术支持的联系地址与电话是：

北京睿智融科控股有限公司

地址：北京市朝阳区望京绿地中心中国锦29层  
邮编：100102  
电话：010-82525486  
QQ：2623717672  
Email： [ThinkTrader@ThinkTrader.net](mailto:ThinkTrader@ThinkTrader.net)

**目 录**

**[1. 使用环境 7](#_Toc23650)**

**[1.1.Windows环境 7](#_Toc15325)**

**[1.2.Linux环境 8](#_Toc32106)**

**[1.3.使用方法 8](#_Toc28403)**

**[1.3.1.C++使用说明 8](#_Toc13623)**

**[1.3.2.Java使用说明 9](#_Toc6530)**

**[1.3.3.C# (C++/CLR) 11](#_Toc28752)**

**[2. 接口描述 12](#_Toc9735)**

**[2.1.接口综述 12](#_Toc18157)**

**[2.2.投资经理指令状态变化图 13](#_Toc3272)**

**[2.3.独立交易员指令状态变化图 13](#_Toc13699)**

**[2.4.API发送指令时序图 14](#_Toc12447)**

**[3. 业务与接口对照 14](#_Toc29393)**

**[3.1.业务概述 16](#_Toc5978)**

**[4. 接口开发说明 16](#_Toc5969)**

**[4.1.1.连接和断线重连 16](#_Toc483)**

**[4.1.2.客户端用户登陆 17](#_Toc10554)**

**[4.1.3.客户端用户登出 17](#_Toc19139)**

**[4.1.4.请求资金账号信息 18](#_Toc31085)**

**[4.1.5.请求委托明细 18](#_Toc26059)**

**[4.1.6.通过指令号请求委托明细 18](#_Toc24846)**

**[4.1.7.请求成交明细 19](#_Toc29151)**

**[4.1.8.通过指令号请求成交明细 19](#_Toc11410)**

**[4.1.9.请求持仓明细 19](#_Toc5685)**

**[4.1.10.请求持仓统计 20](#_Toc10375)**

**[4.1.11.请求两融负债 20](#_Toc28303)**

**[4.1.12.请求两融标的 20](#_Toc9577)**

**[4.1.13.请求期权备兑持仓 21](#_Toc9781)**

**[4.1.14.请求期权组合持仓 21](#_Toc13111)**

**[4.1.15.行情数据请求 21](#_Toc8113)**

**[4.1.16.按市场请求行情 22](#_Toc6829)**

**[4.1.17.请求期权行情 22](#_Toc14252)**

**[4.1.18.订阅行情 22](#_Toc2378)**

**[4.1.19.退订行情 23](#_Toc25424)**

**[4.1.20.查询港股汇率 23](#_Toc20109)**

**[4.1.21.查询产品 23](#_Toc12060)**

**[4.1.22.查询日内回转可用持仓统计 24](#_Toc15372)**

**[4.1.23.上传终端信息 24](#_Toc2327)**

**[4.1.24.开启用户自定义定时器 24](#_Toc1874)**

**[4.1.25.停止用户自定义定时器 25](#_Toc22970)**

**[4.1.26.设置用户下单指令冻结选项 25](#_Toc17323)**

**[4.1.27.普通单委托 25](#_Toc18700)**

**[4.1.28.算法单委托 26](#_Toc31553)**

**[4.1.29.组合算法单委托 27](#_Toc14789)**

**[4.1.30.智能算法单委托 28](#_Toc27197)**

**[4.1.31.组合智能算法单委托 29](#_Toc4731)**

**[4.1.32.随机量委托 30](#_Toc13251)**

**[4.1.33.主动算法委托 30](#_Toc27927)**

**[4.1.34.组合普通单委托 31](#_Toc26628)**

**[4.1.35.撤单操作 32](#_Toc18223)**

**[4.1.36.撤委托操作 32](#_Toc11203)**

**[4.1.37.查询两融综合信息 33](#_Toc25082)**

**[4.2.主推接口 33](#_Toc5361)**

**[4.2.1.主推账户登录状态 33](#_Toc14056)**

**[4.2.2.主推账户登录状态 34](#_Toc11608)**

**[4.2.3.资金账号资金信息 35](#_Toc20028)**

**[4.2.4.主推指令状态明细 36](#_Toc21352)**

**[4.2.5.主推委托明细 37](#_Toc28426)**

**[4.2.6.主推成交明细 38](#_Toc12964)**

**[4.2.7.主推委托错误信息 39](#_Toc7302)**

**[4.2.8.主推撤销错误信息 40](#_Toc6211)**

**[4.2.9.产品净值 41](#_Toc12947)**

**[4.3.回调函数综述 41](#_Toc19964)**

**[4.3.1.请求资金账号信息 42](#_Toc17385)**

**[4.3.2.请求资金账号信息 42](#_Toc13337)**

**[4.3.3.请求委托明细 43](#_Toc11855)**

**[4.3.4.请求成交明细 43](#_Toc15802)**

**[4.3.5.请求持仓明细 43](#_Toc16832)**

**[4.3.6.请求持仓统计 43](#_Toc15695)**

**[4.3.7.请求日内回转持仓 44](#_Toc3245)**

**[4.3.8.请求负债合约 44](#_Toc3488)**

**[4.3.9.请求标的 44](#_Toc24080)**

**[4.3.10.请求备兑持仓 44](#_Toc20887)**

**[4.3.11.请求组合持仓 45](#_Toc27754)**

**[4.3.12.请求产品信息 45](#_Toc4075)**

**[4.3.13.请求合约信息 45](#_Toc3306)**

**[4.3.14.请求汇率信息 45](#_Toc21783)**

**[4.3.15.请求行情信息 46](#_Toc7680)**

**[4.3.16.订阅行情回调 46](#_Toc551)**

**[4.3.17.退订行情回调 46](#_Toc28537)**

**[4.3.18.请求两融综合信息 46](#_Toc5853)**

**[5.容错处理 47](#_Toc23702)**

**[6.问题与解答 47](#_Toc31217)**

**[6.1.onOrder(int nRequestId, int orderID, const XtError& error)函数接收到errorId为0，是不是说明委托成交了？ 47](#_Toc12216)**

**[6.2.下单时可选的操作类型有哪些？ 47](#_Toc6280)**

**[6.3.如何对应一个组合单（算法单）与很多委托回调？ 47](#_Toc25755)**

**[6.4.对于组合单（算法单）而言，如何查看已经成交了多少委托？ 48](#_Toc15882)**

**[6.5.撤单时传入OrderID，如何才能知道撤单成功了呢，尤其是对于组合单和算法单? 48](#_Toc17717)**

**[6.6.程序退出时，如何销毁 XtTraderApi\*？ 48](#_Toc20480)**

**[6.7.api是怎么获取行情的？需要填额外的地址么？ 48](#_Toc32429)**

# 使用环境

|  |  |
| --- | --- |
| XtDataType.h | 枚举类型定义 |
| XtDef.h | 动态库dll输出宏定义 |
| XtError.h | 错误类信息 |
| XtStructs.h | Api所需要的结构体定义头文件 |
| XtTraderApi.h | 交易接口头文件 |
| libeay32.dll | XtTraderApi.dll的依赖库 |
| ssleay32.dll | XtTraderApi.dll的依赖库 |
| XtTraderApi.lib | 静态库文件(C++) win下 |
| XtTraderApi.dll | 动态库文件(C++) win下 |
| libXtTraderApi.so | 动态库文件(C++) linux下 |
| XtTraderJavaApi.lib | lib文件(JNI wrapper) |
| XtTraderJavaApi.dll | 动态库(JNI wrappter) |
| XtTraderNetApi.dll | 动态库(C++/CLR) |
| HsFutuSystemInfo.dll | 期货穿透采集信息的依赖库 |
| WinDataCollect.dll | 期货穿透采集信息的依赖库 |

## 1.1.Windows环境

上述静态库XtTraderApi.dll和动态库XtTraderApi.dll 编译环境为 Windows 7 professional和Visual Studio 2008 sp1。Visual Studio 2008中编译选项为“Win32 Release”。

理论上，vs2008、vs2010、vs2012、vs2013在Windows 7(win32, x64), Windows 8(win32, x64) 均能使用静态库XtTraderApi.dll和动态库XtTraderApi.dll。经过测试已被证明可以正常使用的平台组合有：

1）visual studio 2008和windows 7 x64

2）visual studio 2010和windows 7 x32

## 1.2.Linux环境

Boost版本(1.4.8)

G++版本（gcc-4\_5-branch revision）

Linux内核版本(2.6.34\_x86\_64\_GNU/Linux)

目前支持suse、centos、ubuntu版本。

## 1.3.使用方法

### 1.3.1.C++使用说明

方式一：在demo基础上直接调用

将demo解压后，打开和自身VS版本相适应的XtTraderApiDemo.sln文件，直接运行（暂时只支持了VS2017以及之后版本）。

方式二：将现有API加入调用者原有工程

#### ♦ 总体目录如下

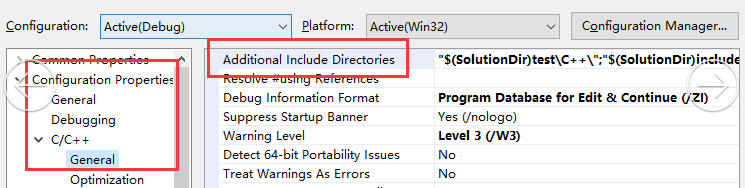
#### 

♦ 加载源文件

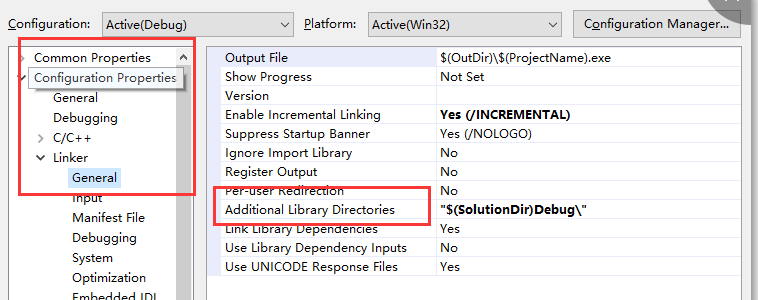
1. 新建一个C++工程。设置工程属性"Configuration" 为Debug，"Platform" 为Win32。
2. 将include目录下的所有头文件导入到工程的Header Files里。
3. 将test/C++目录下的所有头文件(h)导入到工程的Header Files里，将该目录下的源文件(cpp)都导入到工程的Source Files里。
4. 将lib/debug目录下的XtTraderApi.dll和XtTraderApi.lib加载到工程的Resource Files里。
5. 将libeay32.dll、ssleay32.dll放在XtTraderApi.dll同一文件夹下

♦ **设置工程属性**

1. 右键工程名，点击属性(Properties) -> Configuration Properties。
2. C/C++ -> General -> Additional Include Directories 设置头文件路径设置<该路径是您的头文件所在的路径>。见下图



1. Linker -> General-> Additional Library Directories设置**X**tTraderApi.dll和**X**tTraderApi.lib的路径位置，之后确认即可。



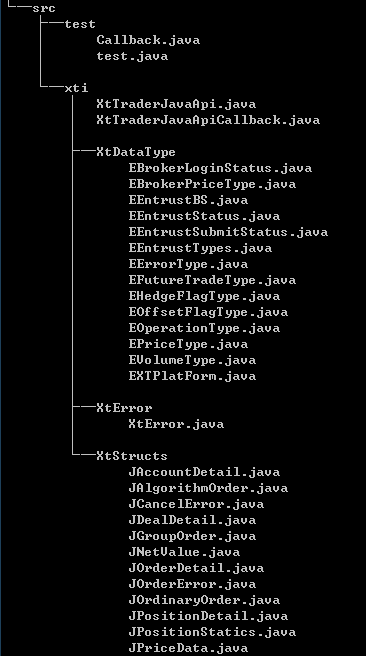
♦ **编译**

右键工程 -> build 即可。

### 1.3.2.Java使用说明

Java接口采用jni技术实现。C++中每个Enum/struct/class与java文件一一对应，C++中命名空间(namespace) xti与Java中包名(package) xti一致，因此不建议修改xti中的代码。测试用例见 traderApiDll/test/Java。

**♦ 目录结构如下**

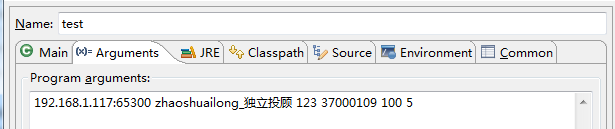


**♦ 建立java工程**

1. 将lib和dll文件需要放到工程根目录下（对于测试用例是 traderApiDll/test/Java/）。
2. Eclipse中导入工程，配置工程参数（见“输入”）。
3. 运行test.java。

♦ 输入运行参数

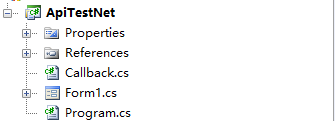
main函数需要输入5个参数，分别为交易服务器地址、用户名、密码、下单账号、每秒下单量、秒数。账号登录成功以后，启动一个后台线程下普通单。 Eclipse中，如下图的参数，表示 37000109登录成功以后开始计时，每秒钟报出100个普通单，报5秒。



### 1.3.3.C# (C++/CLR)

C#接口采用C++/CLR技术实现。使用traderapi时，只需要把XtTraderNetApi.dll加入到 项目References中，不需要直接使用C#版本的头文件。由于头文件提供了C#类中多个数据结构的定义，这里作为参考资料提供出来(traderApiDll/test/C#/header)。

♦ 代码目录如下



**♦ 建立C#工程**

1. 新建一个C# 工程，步骤为：File -> New->Project ->Visual C#->Windows Forms Application；将 Callback.cs 和 Program.cs 导入到工程中。
2. 设置工程属性"Platform target"为x86；具体步骤为：右键工程 -> 属性 -> build -> Platform target -> x86。
3. 将XtTraderNetApi.dll导入工程依赖。选中"References"🡪 "Add Reference…"。
4. 修改参数，编译运行。

# 接口描述

## 2.1.接口综述

1. 前提：创建与服务器的连接。
2. 客户端用户登录：连接成功后，使用userLogin登录用户。
3. 服务器主推用户登录状态：通过 onRtnLoginStatus 可以获得资金账户登录状态。注意：相同用户的登录会导致已登录用户再次接受到 onRtnLoginStatus 主推，资金账户登录状态变化时，也会接收到改主推。
4. 查询和下单操作：可以对登录成功的资金账户进行查询和下单操作。

Api对查询和下单及撤单等请求式操作，使用一问一答的异步回调机制，每个请求都有相应的回调函数，回调函数中可通过nRequestID来找到其请求函数。请求函数均为纯虚函数，使用API时需要在其派生类中实现。这些请求函数包括（函数声明参考XtTraderApi.h）。

## 2.2.投资经理指令状态变化图

审批中

运行中

撤销中

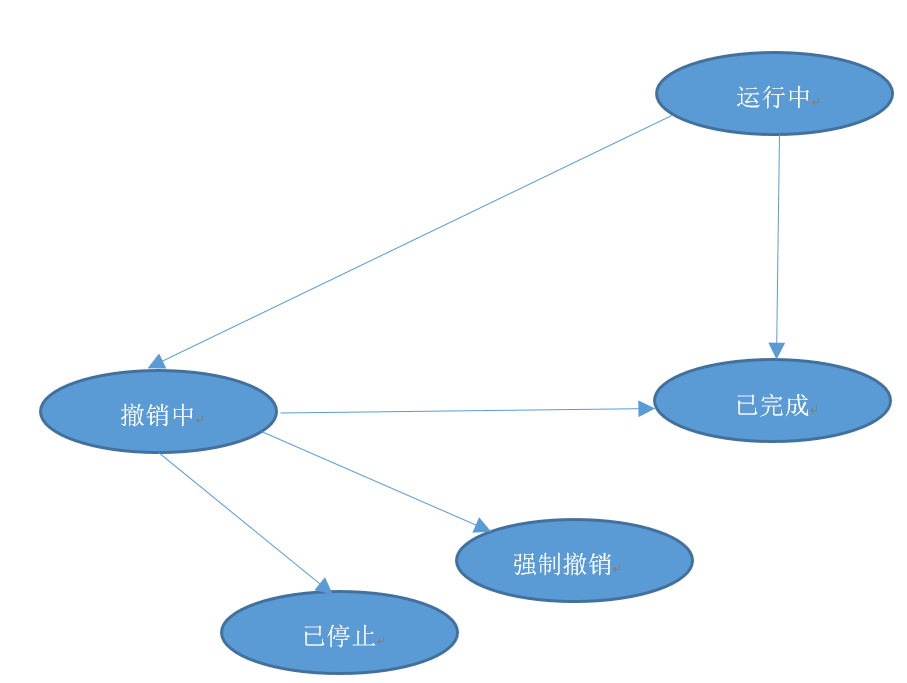
已完成

已停止

已驳回

强制撤销

## 2.3.独立交易员指令状态变化图



## 2.4.API发送指令时序图



# 业务与接口对照

|  |  |  |  |
| --- | --- | --- | --- |
| 业务类型 | 业务 | 请求接口  作用域:XtTraderApi | 响应接口  作用域:XtTraderApiCallback |
| 登陆 | 登陆 | userLogin | onUserLogin |
| 登出 | 登出 | userLogout | onUserLogout |
| 上传终端ctp采集信息 | 上传终端ctp采集信息 | registerUserSystemInfo | None |
| 设置用户自己的定时器 | 设置用户自己的定时器 | startTimer | None |
| 关闭定时器 | 关闭定时器 | stopTimer | None |
| 设置用户下单指令冻结选项 | 设置用户下单指令冻结选项 | setCmdFrzCheckOption | None |
| 交易 | 报单 | order | onOrder |
| 撤指令 | cancel | onCancel |
| 撤委托 | cancelOrder | onCancelOrder |
| 主推 | 用户登录状态 | None | onRtnLoginStatus |
| 用户登录状态 | None | onRtnLoginStatusWithActKey |
| 资金账号信息 | None | onRtnAccountDetail |
| 主推指令明细 | None | onRtnOrder |
| 委托明细 | None | onRtnOrderDetail |
| 主推成交明细 | None | onRtnDealDetail |
| 委托错误信息 | None | onRtnOrderError |
| 撤单信息 | None | onRtnCancelError |
| 产品净值 | None | onRtnNetValue |
| 主推行情数据 | None | onRtnPriceData |
| 定时器回调 | None | onCustomTimer |
| 信用资金账号信息 | None | onRtnCreditAccountDetail |
| 查询 | 资金账号信息 | reqAccountDetail | onReqAccountDetail  onReqCreditAccountDetail |
| 委托明细 | reqOrderDetail | onReqAccountDetail |
| 成交明细 | reqDealDetail | onReqDealDetail |
| 持仓明细 | reqPositionDetail | onReqPositionDetail |
| 持仓统计 | reqPositionStatics | onReqPositionStatics |
| 负债合约:信用 | reqStkcompacts | onReqStkcompacts |
| 查询标的:信用 | reqStksubjects | onReqStksubjects |
| 备兑持仓:期权 | reqCoveredStockPosition | onReqCoveredStockPosition |
| 查询行情 | reqPriceData | onReqPriceData |
| 查询期权组合持仓 | reqStkOptCombPositionDetail | onReqStkOptCombPositionDetail |
| 按市场查询行情 | reqPriceDataByMarket | onReqCInstrumentDetail |
| 查询期权行情数据 | reqInstrumentDetail | onReqCInstrumentDetail |
| 查询汇率 | reqGGTReferenceRate | onReqReferenceRate |
| 订阅主推行情数据 | subscribQuote | onSubscribQuote |
| 退订主推行情数据 | unSubscribQuote | onUnSubscribQuote |
| 请求可用统计 | queryRevolvePositions | onReqRevolvePositions |
| 查询两融综合信息 | reqCreditDetail | onReqCreditDetail |
| 查询产品 | reqProductData | onReqProductData |

## 3.1.业务概述

交易类和主推类接口会相互关联，如用户通过order报单<指令>，马上就会收到指令回调onOrder，说明迅投系统已经收到该报指令，并生成相应的委托，迅投系统会把该委托报给柜台。报单进入柜台之后，如果报单的交易状态发生变化，就会收到报单的主推回报onRtnOrderDetail；如果报单被撮合(部分)成交，就会收到成交回报onRtnDealDetail。

# 接口开发说明

## 4.1.1.连接和断线重连

**void onConnected(bool success, const char\*** **errorMsg)**

参数说明：

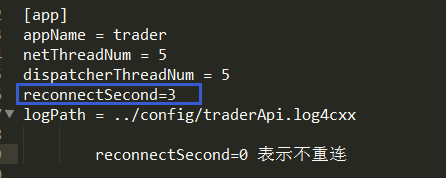
success 连接是否成功；

errorMsg 连接失败时，返回失败的原因。

创建 XtTraderApi实例以后，在init()中尝试连接服务器，不管成功或失败，都会通过onConnected返回连接状态。

在使用XtTraderApi过程中，如果出现断开，会通过onConnected推送连接断开的信息。XtTraderApi默认支持断线重连，重连时间为3s。如果您想关闭断线重连或者改变重连时间，可以在traderApi.ini中设置。../config/XtTraderApi.log4cxx对应的当前路径是应用程序启动的路径。

对于使用Java和C#接口，保证 XtTraderApi.dll 与 config/XtTraderApi.ini的相对位置，也可以断线重连。



以下函数为XtTraderApi中的接口说明：

## 4.1.2.客户端用户登陆

**virtual void userLogin(const char\* userName, const char\* password, int nRequestId, const char\* machineInfo = NULL, const char\* appid = NULL, const char\* authcode = NULL)**

参数说明：

userName： 管理端中创建的用户。

password：管理端中相应的密码。

nRequestId：需要客户维护的请求顺序ID。

machineInfo：登陆传入的终端信息，用于作为后续委托时的站点。

appid ：期货ip穿透在柜台注册的appid，可以送空，使用api默认值。

authcode：期货ip穿透注册的authcode，可以送空，使用api默认值。

## 4.1.3.客户端用户登出

**virtual void userLogout(const char\* userName, const char\* password, int nRequestId)**

参数说明：

userName： 管理端中创建的用户。

password：管理端中相应的密码。

nRequestId：需要客户维护的请求顺序ID。

## 4.1.4.请求资金账号信息

**virtual void reqAccountDetail(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.5.请求委托明细

**virtual void reqOrderDetail(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.6.通过指令号请求委托明细

**virtual void reqOrderDetail(const char\* accountID, int nRequestId, int nOrderID, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.7.请求成交明细

**virtual void reqDealDetail(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.8.通过指令号请求成交明细

**virtual void reqDealDetail(const char\* accountID, int nRequestId, int nOrderID, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.9.请求持仓明细

**virtual void reqPositionDetail(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.10.请求持仓统计

**virtual void reqPositionStatics(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.11.请求两融负债

**virtual void reqStkcompacts(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.12.请求两融标的

**virtual void reqStksubjects(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.13.请求期权备兑持仓

**virtual void reqCoveredStockPosition(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.14.请求期权组合持仓

**virtual void reqStkOptCombPositionDetail(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.15.行情数据请求

**void reqPriceData(char\* exchangeId, char\* instrumentId, int nRequestId)**

参数说明：

exchangeId：证卷交易市场定义 代码,如 SH ,SZ。

instrumentId：证券代码。

nRequestId：为客户自己维护的请求顺序ID。

## 4.1.16.按市场请求行情

**virtual void reqPriceDataByMarket(const char\* exchangeId, int nRequestId)**

参数说明：

exchangeId：市场，可以从marketType里面取。

nRequestId：客户自己维护的请求顺序ID。

## 4.1.17.请求期权行情

**virtual void reqInstrumentDetail(const char\* accountID, int nRequestId,const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.18.订阅行情

**virtual void subscribQuote(const CSubscribData\* data, int nRequestId)**

参数说明：

CSubscribData结构说明

struct CSubscribData

{

CSubscribData();

~CSubscribData();

int m\_nPlatformID; //平台ID

char m\_strExchangeID[16]; //市场代码

char m\_strInstrumentID[32]; //合约代码,当其值为allCode时，订阅整个市场

EXTOfferStatus m\_eOfferStatus; //报盘状态

};

nRequestId：客户自己维护的请求顺序ID。

## 4.1.19.退订行情

**virtual void unSubscribQuote(const CSubscribData\* data, int nRequestId)**

参数说明：

CSubscribData结构说明

struct CSubscribData

{

CSubscribData();

~CSubscribData();

int m\_nPlatformID; //平台ID

char m\_strExchangeID[16]; //市场代码

char m\_strInstrumentID[32]; //合约代码,当其值为allCode时，订阅整个市场

EXTOfferStatus m\_eOfferStatus; //报盘状态

};

nRequestId：客户自己维护的请求顺序ID。

## 4.1.20.查询港股汇率

**virtual void reqGGTReferenceRate(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.21.查询产品

**virtual void reqProductData(int nRequestId)**

参数说明：

nRequestId：客户自己维护的请求顺序ID。

## 4.1.22.查询日内回转可用持仓统计

**virtual void queryRevolvePositions(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.23.上传终端信息

**virtual void registerUserSystemInfo(const char\* accountID, const char\* IpPortAddr, const int len, const char\* CTPSystemInfo, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

IpPortAddr：上传终端信息机器的ip和端口信息(格式为ip:port，例如127.0.0.1:58000)。

len：ctp采集信息内容的长度len，即CTP\_GetSystemInfo入参nLen。

CTPSystemInfo：ctp采集到的内容。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.24.开启用户自定义定时器

**virtual void startTimer(int millsec)**

参数说明：

millsec：定时器时间间隔。

## 4.1.25.停止用户自定义定时器

**virtual void stopTimer()**

## 4.1.26.设置用户下单指令冻结选项

**virtual void setCmdFrzCheckOption(int nCmdFrzCheckOption)**

参数说明：

nCmdFrzCheckOption：指令冻结类型，1 禁止 2 警告。

## 4.1.27.普通单委托

**virtual void order(const COrdinaryOrder\* orderInfo, int nRequestId,const char\* accountKey = "")**

参数说明：

COrdinaryOrder 结构说明：

struct COrdinaryOrder

{

COrdinaryOrder();

~COrdinaryOrder();

char m\_strAccountID[32]; // 资金账户ID;

double m\_dPrice; // 指定价，仅在报价方式为PRTP\_FIX(指定价)时有效;

double m\_dSuperPriceRate; // 单笔超价,只对期货有效

int m\_nVolume; // 委托量

char m\_strMarket[32]; // 合约市场

char m\_strInstrument[32]; // 委托合约

EPriceType m\_ePriceType; // 报价方式：指定价、最新价

EOperationType m\_eOperationType; // 下单类型：开、平、买、卖

EHedge\_Flag\_Type m\_eHedgeFlag; // 套利标志

double m\_dOccurBalance; //直接还款的金额 //仅直接还款用

ETimeCondition m\_eTimeCondition; //期货条件单时间条件

EVolumeCondition m\_eVolumeCondition; //期货条件单数量条件

char m\_strSecondInstrument[32]; // 期权组合委托合约

double m\_dSecondPrice; // 期权组合委托价

ESideFlag m\_eFirstSideFlag; //第一腿合约持仓类型

ESideFlag m\_eSecondSideFlag; //第二腿合约持仓类型

char m\_strCombID[32]; // 组合持仓编号，用于解除组合策略

char m\_strRemark[64]; // 投资备注

};

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.28.算法单委托

**virtual void order(const CAlgorithmOrder\* orderInfo, int nRequestId,const char\* accountKey = "")**

参数说明：

CAlgorithmOrder 结构说明:

struct CAlgorithmOrder

{

CAlgorithmOrder();

~CAlgorithmOrder();

char m\_strAccountID[32]; // 资金账号

char m\_strMarket[32]; // 市场

char m\_strInstrument[32]; // 合约代码

double m\_dPrice; // 报价

double m\_dSuperPriceRate; // 单笔超价

int m\_nSuperPriceStart; // 超价起始笔数

double m\_dPriceRangeRate; // 波动区间

int m\_nVolume; // 委托量

double m\_dSingleVolumeRate; // 单比下单比率

int m\_nSingleNumMin; // 单比下单量最小值

int m\_nSingleNumMax; // 单比下单量最大值

int m\_nLastVolumeMin; // 尾单最小量

double m\_dPlaceOrderInterval; // 下单间隔

double m\_dWithdrawOrderInterval; // 撤单间隔

int m\_nMaxOrderCount; // 最大下单次数

int m\_nValidTimeStart; // 有效开始时间 来自股票，待定

int m\_nValidTimeEnd; // 有效结束时间 来自股票，待定

EOperationType m\_eOperationType; // 下单操作：开平、多空…

EVolumeType m\_eSingleVolumeType; // 单笔下单基准

EPriceType m\_ePriceType; // 报价方式：对手、最新…

EHedgeFlagType m\_eHedgeFlag; // 套利标志

EXtOverFreqOrderMode m\_eOverFreqOrderMode; //委托频率过快时的处理方式

ETimeCondition m\_eTimeCondition; //期货条件单时间条件

EVolumeCondition m\_eVolumeCondition; //期货条件单数量条件

char m\_strRemark[64]; // 投资备注

};

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.29.组合算法单委托

**virtual void order(const CGroupOrder\* orderInfo, int nRequestId,const char\* accountKey = "")**

参数说明：

CGroupOrder 结构说明:

struct CGroupOrder

{

CGroupOrder();

~CGroupOrder();

CAlgorithmOrder m\_orderParam; // 下单配置

char m\_strMarket[500][32]; // 市场列表

char m\_strInstrument[500][32]; // 证券代码

int m\_nVolume[500]; // 每只股票的下单量

int m\_nOrderNum; // 股票报单数

char m\_strRemark[64]; // 投资备注

};

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

注意：组合单只支持股票，不支持期货，一次最多可下500只股票。

## 4.1.30.智能算法单委托

**virtual void order(const CIntelligentAlgorithmOrder\* orderInfo, int nRequestId,const char\* accountKey = "")**

参数说明：

CIntelligentAlgorithmOrder结构说明:

struct CIntelligentAlgorithmOrder

{

CIntelligentAlgorithmOrder();

~CIntelligentAlgorithmOrder();

char m\_strAccountID[32]; //账号

char m\_strMarket[32]; //市场

char m\_strInstrument[32]; //合约

char m\_strOrderType[32]; //算法名称，VWAP，TWAP，VP，PINLINE，FLOAT

double m\_dPrice; //基准价

int m\_nVolume; //下单总量

int m\_nValidTimeStart; //有效开始时间

int m\_nValidTimeEnd; //有效结束时间

double m\_dMaxPartRate; // 量比比例, 用户设定, 当MaxPartRate==100%, 表示没有限制

double m\_dMinAmountPerOrder; //委托最小金额

EOperationType m\_eOperationType; //下单操作：买入卖出

EPriceType m\_ePriceType; //报价方式：市价和指定价

char m\_strRemark[64]; // 投资备注

};

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.31.组合智能算法单委托

**virtual void order(const CAlgGroupOrder\* orderInfo, int nRequestId,const char\* accountKey = "")**

参数说明：

CAlgGroupOrder结构说明:

struct CAlgGroupOrder

{

CAlgGroupOrder();

~CAlgGroupOrder();

CIntelligentAlgorithmOrder m\_orderParam; //下单配置

char m\_strMarket[500][32]; //市场列表

char m\_strInstrument[500][32]; //证券代码

int m\_nVolume[500]; //每只股票的下单量

int m\_nOrderNum; //股票只数

char m\_strRemark[64]; // 投资备注

};

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

注意：组合单只支持股票，一次最多可下500只股票。

## 4.1.32.随机量委托

**virtual void order(const CRandomOrder\* orderInfo, int nRequestId,const char\* accountKey = "")**

参数说明：

CRandomOrder结构说明:

struct CRandomOrder

{

CRandomOrder();

~CRandomOrder();

char m\_strAccountID[32]; //账号

char m\_strMarket[32]; //市场

char m\_strInstrument[32]; //合约

double m\_dPrice; //基准价

int m\_nVolume; //下单总量

int m\_nValidTimeElapse; //下单间隔

int m\_nSingleNumMin; //单比下单量最小值

int m\_nSingleNumMax; //单比下单量最大值

EOperationType m\_eOperationType; //下单操作：开平、多空

EPriceType m\_ePriceType; //报价方式：对手、最新

char m\_strRemark[64]; // 投资备注

};

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.33.主动算法委托

**virtual void order(const CExternAlgorithmOrder\* orderInfo, int nRequestId,const char\* accountKey = "")**

参数说明：

CExternAlgorithmOrder结构说明:

struct CExternAlgorithmOrder

{

CExternAlgorithmOrder();

~CExternAlgorithmOrder();

char m\_strAccountID[32]; //账号

char m\_strMarket[32]; //市场

char m\_strInstrument[32]; //合约

char m\_strOrderType[32]; //算法名称，FTAIWAP，ALGOINTERFACE, ZEUS....

double m\_dPrice; //基准价

int m\_nVolume; //下单总量

int m\_nValidTimeStart; //有效开始时间

int m\_nValidTimeEnd; //有效结束时间

double m\_dMaxPartRate; // 量比比例, 用户设定, 当MaxPartRate==100%, 表示没有限制

double m\_dMinAmountPerOrder; //委托最小金额

EOperationType m\_eOperationType; //下单操作：买入卖出

EPriceType m\_ePriceType; //报价方式：市价和指定价

char m\_strRemark[64]; // 投资备注

};

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.34.组合普通单委托

**virtual void order(const COrdinaryGroupOrder\* orderInfo, int nRequestId,const char\* accountKey = "")**

参数说明：

COrdinaryGroupOrder结构说明:

struct COrdinaryGroupOrder

{

COrdinaryGroupOrder();

~COrdinaryGroupOrder();

char m\_strAccountID[32]; // 资金账户ID，如果为子账户，则为子账户ID

double m\_dSuperPriceRate; // 单笔超价百分比

EPriceType m\_ePriceType; // 报价方式： 指定价，最新价 对手价……

EHedgeFlagType m\_eHedgeFlag; // 套利标志

EXtOverFreqOrderMode m\_eOverFreqOrderMode; //委托频率过快时的处理方式

ETimeCondition m\_eTimeCondition; //期货条件单时间条件

EVolumeCondition m\_eVolumeCondition; //期货条件单数量条件

char m\_strMarket[500][32]; //市场列表

char m\_strInstrument[500][32]; //证券代码

int m\_nVolume[500]; //每只股票的下单量

double m\_dPrice[500]; //每只股票的下单价格

EOperationType m\_eOperationType[500]; //每只股票的下单类型

int m\_nOrderNum; //股票只数

char m\_strRemark[64]; // 投资备注

};

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

注意：组合单只支持股票，一次最多可下500只股票。

## 4.1.35.撤单操作

**virtual void cancel(int orderID, int nRequestId)**

参数说明：

orderID：指令ID，由服务器返回，onOrder的orderID, onRtnOrder的data->m\_nOrderID，不是单笔委托的委托号(m\_strOrderSysID)。

nRequestId：请求顺序ID，用户自己分配。

注意：如果第一次撤单后，指令状态一直是”撤销中“，第二次撤单时XtTraderApi会强制终止该下单指令（避免撤不掉单的问题）。

## 4.1.36.撤委托操作

**virtual void cancelOrder(const char\* accountID, const char\* orderSyeId, const char\* exchangeId, const char\* instrumentId, int nRequestId,const char\* accountKey = "")**

参数说明：

accountID： 用户绑定的资金账号；

orderSysId：委托号，该委托号为onOrderDetailDetail返回。

orderID：指令ID，由迅投服务器返回，api的onOrder接口返回。

exchangeId：市场代码。

instrumentId：合约代码。

nRequestId：请求号ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.1.37.查询两融综合信息

**virtual void reqCreditDetail(const char\* accountID, int nRequestId, const char\* accountKey = "")**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

accountKey：当资金账号ID相同时，用于区分两个账号的标识，在onRtnLoginStatusWithActKey（）里面推送。

## 4.2.主推接口

以下函数为XtTraderApiCallback中的接口：

## 4.2.1.主推账户登录状态

**virtual void onRtnLoginStatus(const char\* accountID, EBrokerLoginStatus status, const char\* errorMsg)**

参数说明：

accountID：用户绑定下的资金账号。

enum EBrokerLoginStatus

{

BROKER\_LOGIN\_STATUS\_INALID = -1, //枚举起始值，不必关注

BROKER\_LOGIN\_STATUS\_OK = 0, //初始化完成，账户可用

BROKER\_LOGIN\_STATUS\_WAITING\_LOGIN,//连接中

BROKER\_LOGIN\_STATUS\_LOGINING, //登录中

BROKER\_LOGIN\_STATUS\_FAIL, //登录失败

BROKER\_LOGIN\_STATUS\_INITING, //初始化中

BROKER\_LOGIN\_STATUS\_CORRECTING, //数据刷新校正中

BROKER\_LOGIN\_STATUS\_CLOSED, //休市中(盘后状态)

};

errorMsg：如果登陆成功，errorMsg为空；如果没登陆成功，通过errorMsg返回具体错误信息。

注意：登录状态为 BROKER\_LOGIN\_STATUS\_OK时，才可以下单。

## 4.2.2.主推账户登录状态

**virtual void onRtnLoginStatusWithActKey(const char\* accountID, EBrokerLoginStatus status, int brokerType, const char\* accountKey, const char\* errorMsg)**

参数说明：

accountID：用户绑定下的资金账号。

enum EBrokerLoginStatus

{

BROKER\_LOGIN\_STATUS\_INALID = -1, //枚举起始值，不必关注

BROKER\_LOGIN\_STATUS\_OK = 0, //初始化完成，账户可用

BROKER\_LOGIN\_STATUS\_WAITING\_LOGIN,//连接中

BROKER\_LOGIN\_STATUS\_LOGINING, //登录中

BROKER\_LOGIN\_STATUS\_FAIL, //登录失败

BROKER\_LOGIN\_STATUS\_INITING, //初始化中

BROKER\_LOGIN\_STATUS\_CORRECTING, //数据刷新校正中

BROKER\_LOGIN\_STATUS\_CLOSED, //休市中(盘后状态)

};

errorMsg：如果登陆成功，errorMsg为空；如果没登陆成功，通过errorMsg返回具体错误信息。

accountKey：当资金账号ID相同时，用于区分两个账号的标识

注意：登录状态为 BROKER\_LOGIN\_STATUS\_OK时，才可以下单。

## 4.2.3.资金账号资金信息

**virtual void onRtnAccountDetail(const char\* accountID, const CAccountDetail\* data)**

参数说明：

CAccountDetail 结构说明：

struct CAccountDetail

{

CAccountDetail();

~CAccountDetail();

char m\_strAccountID[32]; // 资金账号

char m\_strStatus[32]; // 资金账号的状态

char m\_strTradingDate[16]; // 交易日

double m\_dFrozenMargin; // 期货冻结保证金，股票冻结资金

double m\_dFrozenCash; // 冻结保证金、冻结手续费之和

double m\_dFrozenCommission; // 冻结手续费

double m\_dRisk; // 期货风险度

double m\_dPreBalance; // 期货期初权益

double m\_dBalance; // 总资产、期货动态权益

double m\_dAvailable; // 可用资金

double m\_dCommission; // 已用手续费

double m\_dPositionProfit; // 持仓盈亏

double m\_dCloseProfit; // 平仓盈亏

double m\_dCurrMargin; // 当前占用保证金

double m\_dInstrumentValue; // 合约价值

double m\_dDeposit; // 期货入金

double m\_dWithdraw; // 期货出金

double m\_dCredit; // 信用额度

double m\_dMortgage; // 质押

double m\_dTotalDebit; // 总负债

double m\_dPremiumNetExpense; // 权利金净支出 用于股票期权

double m\_dEnableMargin; // 可用保证金 用于股票期权

};

注意：一旦持仓股票或期货价格发生变动，账号资产就会发生变化，就会有这个主推。该数据变更频繁，如果对下游造成性能压力，可以不实现对应接口，而只在需要时进行查询。

## 4.2.4.主推指令状态明细

**virtual void onRtnOrder(const COrderInfo\* data)**

用处：获取主推的指令明细、指令状态。

参数说明：

COrderInfo结构说明:

struct COrderInfo

{

COrderInfo();

~COrderInfo();

char m\_strAccountID[32]; //资金账号

int m\_nAccountType; //账号类型

int m\_nOrderID[32]; //指令ID

int m\_startTime[16]; //下达时间

int m\_endTime[32]; //结束时间

int m\_eStatus[32]; //状态

int m\_dTradedVolume[32]; //成交量

int m\_strMsg; //指令执行信息

int m\_canceler[32]; //撤销者

EXTBrokerType m\_eBrokerType; //账号类型

char m\_strRemark[64]; // 投资备注

};

## 4.2.5.主推委托明细

**virtual void onRtnOrderDetail(const COrderDetail\* data)**

用处：获取主推的报单和委托明细。

普通单、算法单： 一个单子（指令）对应一笔委托，推送一次。

组合单：一个单子可能对应多笔委托，分多次推送。

参数说明：

COrderDetail结构说明:

struct COrderDetail

{

COrderDetail();

~COrderDetail();

char m\_strAccountID[32]; // 资金账号

char m\_strExchangeID[16]; //交易所代码

char m\_strProductID[32]; //合约品种

char m\_strInstrumentID[32]; //合约代码

double m\_dLimitPrice; //限价单的限价，就是报价

char m\_strOrderSysID[32]; //委托号

int m\_nTradedVolume; //已成交量

int m\_nTotalVolume; //当前总委托量

double m\_dFrozenMargin; //冻结保证金

double m\_dFrozenCommission; //冻结手续费

double m\_dAveragePrice; //成交均价

double m\_dTradeAmount; //成交额 期货=均价\*量\*合约乘数

int m\_nErrorID; //错误号

char m\_strErrorMsg[256]; //错误信息

char m\_strInsertDate[16]; //日期

char m\_strInsertTime[32]; //时间

int m\_nOrderID; //下单ID(指令号)

EBrokerPriceType m\_nOrderPriceType; //报单类型，例如市价单、限价单、最新价

EEntrustBS m\_nDirection; //期货多空

EOffsetFlagType m\_eOffsetFlag; //期货开平 股票买卖

EHedgeFlagType m\_eHedgeFlag; //投机 套利 套保

EEntrustSubmitStatus m\_eOrderSubmitStatus; //提交状态

EEntrustStatus m\_eOrderStatus; //委托状态

EEntrustTypes m\_eEntrustType; //委托类别

ECoveredFlag m\_eCoveredFlag; //备兑标记 '0' - 非备兑，'1' - 备兑

char m\_strRemark[64]; // 投资备注

};

注意：产生对敲时，服务器端没有委托反馈，故onRtnOrder中无对应的委托明细。onOrder 一般会在 指令状态修改为 OCS\_APPROVING（审批中）后得到反馈，而不是onRtnOrder之前。

## 4.2.6.主推成交明细

**virtual void onRtnDealDetail(const CDealDetail\* data)**

获取主推的成交明细。onRtnDealDetail一般迟于onRtnOrderDetail推送，即便报单已经结束，onRtnDealDetail仍然可能继续推送。

参数说明:

struct CDealDetail

{

CDealDetail();

~CDealDetail();

char m\_strAccountID[32]; //资金账号

char m\_strExchangeID[16]; //交易所代码

char m\_strProductID[32]; //合约品种

cha r m\_strInstrumentID[32]; //合约代码

char m\_strTradeID[32]; //成交编号

char m\_strOrderSysID[32]; //委托编号

double m\_dAveragePrice; //成交均价

int m\_nVolume; //成交量 期货单位手 股票做到股

char m\_strTradeDate[16]; //成交日期

char m\_strTradeTime[32]; //成交时间

double m\_dComssion; //手续费

double m\_dAmount; //成交额 期货=均价\*量\*合约乘数

int m\_nOrderID; //下单ID(指令ID)

EEntrustBS m\_nDirection; //期货多空

EOffsetFlagType m\_nOffsetFlag; //期货开平 股票买卖

EHedgeFlagType m\_nHedgeFlag; //投机 套利 套保

EBrokerPriceType m\_nOrderPriceType; //类型，例如市价单 限价单

EEntrustTypes m\_eEntrustType; //委托类别

ECoveredFlag m\_eCoveredFlag; //备兑标记 '0' - 非备兑，'1' - 备兑

char m\_strRemark[64]; // 投资备注

};

## 4.2.7.主推委托错误信息

**virtual void onRtnOrderError(const COrderError\* data)**

委托请求被迅投风控打回，或者柜台打回，均通过此接口推送。如股票单笔下单量不是100的整数倍；对敲单的错误信息也会通过这个接口推送。

参数说明：

struct COrderError

{

CDealDetail();

~CDealDetail();

char m\_strAccountID[32]; //资金账号

int m\_nErrorID; //错误号

char m\_strErrorMsg[32]; //错误信息

int m\_nRequestID; //请求号

int m\_nOrderID[32]; //指令号

};

*注意：对于这类情况，XtTraderApi不会自动撤销原先的报单（指令）*

## 4.2.8.主推撤销错误信息

**virtual void onRtnCancelError(const CCancelError\* data)**

获取主推的撤委托错误信息。撤委托请求被迅投系统风控打回，或者撤销请求被柜台打回，均通过这个接口推送。

参数说明：

struct CCancelError

{

CCancelError();

~CCancelError();

char m\_strAccountID[32]; //资金账号

int m\_nErrorID; //错误号

char m\_strErrorMsg[256]; //错误信息

int m\_nOrderID; //指令号

int m\_nRequestID; //请求号

};

## 4.2.9.产品净值

**virtual void onRtnNetValue(const CNetValue\* data)**

注意：一旦持仓股票或期货价格发生变动，账号资产就会发生变化，就会有此主推。

参数说明：

struct CNetValue

{

CNetValue();

~CNetValue();

int m\_nProductId; //迅投产品ID

int m\_nTypes; //产品类型 1-普通基金 2-分级基金

double m\_dTotalNetValue; //产品净资产, 产品净值

double m\_dNetValue; //母基金单位净值

double m\_dBNetValue; //B级基金单位净值

int m\_nUpdateTime; //更新时间

};

## 4.3.回调函数综述

对应这些请求函数的回调函数遵循统一的命名规则，即“on[请求函数名]”，“[请求函数名]” 首字母大写。例如，userLogin 对应的回调函数为onUserLogin，reqAccountDetail 对应 onReqAccountDetail。回调函数实现为不做任何操作，使用时根据需要在派生类中实现（测试用例）。

Api 对委托状态变化和下单导致的成交和资金状态的变化等状态变化信息使用主推方式实时向客户主推。主推接口的格式为 “onRtn[主推信息]”。例如 onRtnDealDetail 是主推成交明细信息。

注意事项：

1. onOrder中返回成功，仅仅表示下单指令已发出，并不意味着委托成功。以onRtnOrderDetail推送的指令报单状态为准。交易成功后，服务器会主推 onRtnOrderDetail（委托明细）和 onRtnDealDetail（成交明细）。
2. onCancel中返回成功，仅仅表示撤单指令已发出，并不意味着未成委托撤成功。以onRtnOrderDetail推送的指令报单状态为准。

## 4.3.1.请求资金账号信息

**virtual void onReqAccountDetail(const char\* accountID, int nRequestId, const CAccountDetail\* data, bool isLast, const XtError& error)**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

CAccountDetail：资金账号明细(见onRtnAccountDetail)。

isLast：是否为最后一条反馈。

XtError：错误号以及错误信息。

## 4.3.2.请求资金账号信息

**virtual void onReqCreditAccountDetail(const char\* accountID, int nRequestId, const CCreditAccountDetail\* data, bool isLast, const XtError& error)**

参数说明：

accountID：资金账号ID。

nRequestId：客户自己维护的请求顺序ID。

CCreditAccountDetail：两融资金账号明细。

isLast：是否为最后一条反馈。

XtError：错误号以及错误信息。

## 4.3.3.请求委托明细

**virtual void onReqOrderDetail(const char\* accountID, int nRequestId, const COrderDetail\* data, bool isLast, const XtError& error)**

COrderDetail：委托明细。

其余参数说明同onReqAccountDetail。

## 4.3.4.请求成交明细

**virtual void onReqDealDetail(const char\* accountID, int nRequestId, const CDealDetail\* data, bool isLast, const XtError& error)**

CDealDetail：成交明细。

其余参数说明同onReqAccountDetail。

## 4.3.5.请求持仓明细

**virtual void onReqPositionDetail(const char\* accountID, int nRequestId, const CPositionDetail\* data, bool isLast, const XtError& error)**

CPositionDetail：持仓明细。

其余参数说明同onReqAccountDetail。

## 4.3.6.请求持仓统计

**virtual void onReqPositionStatics(const char\* accountID, int nRequestId, const CPositionStatics\* data, bool isLast, const XtError& error)**

CPositionStatics：持仓统计。

其余参数说明同onReqAccountDetail。

## 4.3.7.请求日内回转持仓

**virtual void onReqRevolvePositions(const char\* accountID, int nRequestId, const CPFPositionStatics\* data, bool isLast, const XtError& error)**

CPFPositionStatics：日内回转持仓统计。

其余参数说明同onReqAccountDetail。

## 4.3.8.请求负债合约

**virtual void onReqStkcompacts(const char\* accountID, int nRequestId, const CStkCompacts\* data, bool isLast, const XtError& error)**

CStkCompacts：两融负债合约。

其余参数说明同onReqAccountDetail。

## 4.3.9.请求标的

**virtual void onReqStksubjects(const char\* accountID, int nRequestId, const CStkSubjects\* data, bool isLast, const XtError& error)**

CStkSubjects：两融标的。

其余参数说明同onReqAccountDetail。

## 4.3.10.请求备兑持仓

**virtual void onReqCoveredStockPosition(const char\* accountID, int nRequestId, const CCoveredStockPosition\* data, bool isLast, const XtError& error)**

CCoveredStockPosition：期权备兑持仓。

其余参数说明同onReqAccountDetail。

## 4.3.11.请求组合持仓

**virtual void onReqStkOptCombPositionDetail(const char\* accountID, int nRequestId, const CStockOptionCombPositionDetail\* data, bool isLast, const XtError& error)**

CStockOptionCombPositionDetail：期权组合持仓。

其余参数说明同onReqAccountDetail。

## 4.3.12.请求产品信息

**virtual void onReqProductData(int nRequestId, const CProductData\* data, bool isLast, const XtError& error)**

CProductData：产品信息。

其余参数说明同onReqAccountDetail。

## 4.3.13.请求合约信息

**virtual void onReqCInstrumentDetail(const char\* accountID, int nRequestId, const CInstrumentDetail\* data, bool isLast, const XtError& error)**

CInstrumentDetail：合约信息。

其余参数说明同onReqAccountDetail。

## 4.3.14.请求汇率信息

**virtual void onReqReferenceRate(const char\* accountID, int nRequestId, const CReferenceRate\* data, bool isLast, const XtError& error)**

CReferenceRate：港股汇率信息。

其余参数说明同onReqAccountDetail。

## 4.3.15.请求行情信息

**virtual void onReqPriceData(int nRequestId, const CPriceData\* data, const XtError& error)**

CPriceData：行情信息。

nRequestId：客户自己维护的请求顺序ID。

XtError：错误号以及错误信息。

## 4.3.16.订阅行情回调

**virtual void onSubscribQuote(int nRequestId, const CSubscribData\* data, const XtError& error)**

CSubscribData：订阅行情信息。

nRequestId：客户自己维护的请求顺序ID。

XtError：错误号以及错误信息。

## 4.3.17.退订行情回调

**virtual void onUnSubscribQuote(int nRequestId, const CSubscribData\* data, const XtError& error)**

CSubscribData：订阅行情信息。

nRequestId：客户自己维护的请求顺序ID。

XtError：错误号以及错误信息

## 4.3.18.请求两融综合信息

**virtual void onReqCreditDetail(const char\* accountID, int nRequestId, const CCreditDetail\* data, bool isLast, const XtError& error)**

CCreditDetail：两融综合信息。

其余参数说明同onReqAccountDetail。

# 5.容错处理

针对一些极端情况，我们进行了特殊处理以保证交易的正常进行。

XtTraderApi重启时，如果有未完成的单子。XtTraderApi用户登录收到账号主推后，XtTraderApi会恢复未完成的单子，并通过onRtnOrderDetail推送。对于用户离线时，执行结束的单子，XtTraderApi不予推送，客户可通过 reqDealDetail向服务器请求成交数据。

# 6.问题与解答

## 6.1.onOrder(int nRequestId, int orderID, const XtError& error)函数接收到errorId为0，是不是说明委托成交了？

1. error.ErrorID()!= 0 时,orderID == -1，说明单子参数填写有误。
2. error.ErrorID()==0时，orderID!=-1，只能说明产生了指令，不能说明委托成功。
3. onRtnOrderDetail中m\_nErrorID为0说明委托成功，onRtnDealDetail中返回成交明细。
4. 超出指令后剩余量0，这个是风控返回，应该是没可用资金的原因。
5. 详细说明见常见错误文档

## 6.2.下单时可选的操作类型有哪些？

在COrdinaryOrder、CAlgorithmOrder和CGroupOrder中都有一个字段m\_eOperationType，设置这个值即可。具体请参考XtDataType.h文件EOperationType字段。

## 6.3.如何对应一个组合单（算法单）与很多委托回调？

下单时是不存在OrderID的。OrderID是在下单后由迅投服务器创建并返回给XtTraderApiCallback中onOrder函数。报单（指令）状态发生变化时，推送onRtnOrder。委托状态发生变化时，服务器主推委托明细(onRtnOrderDetail)和成交明细(onRtnDealDetail)。COrderDetail 和 CDealDetail中都有一个字段m\_nOrderID，通过这个字段可以建立单子与委托的对应关系。

## 6.4.对于组合单（算法单）而言，如何查看已经成交了多少委托？

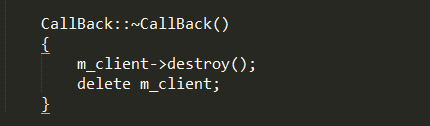
有两种方法。方法一，通过 reqOrderDetail接口主动查询某个报单（指令）的所有委托明细；方法二，报单后收集onRtnOrderDetail，缓存起来，统计委托成交，保单（指令）结束后，释放缓存。

## 6.5.撤单时传入OrderID，如何才能知道撤单成功了呢，尤其是对于组合单和算法单?

cancel函数进行撤单。onCancel函数接收撤单成功或者失败的消息，onCancel接收到 error.isSuccess()表示撤单指令发送成功。以onRtnOrderDetail或者onReqOrderDetail推送的指令报单状态为准。

## 6.6.程序退出时，如何销毁 XtTraderApi\*？

一般使用XtTraderApi\* 对象作为一个成员变量来使用的，退出时需要先调用 destroy()函数，然后 delete 指针。



## 6.7.api是怎么获取行情的？需要填额外的地址么？

目前api连接专门的api服务器apiservice，apiservice的监听端口是65300。注意**api只需要填写这么一个地址**。

Api提供获取行情的接口，叫做reqPriceData。其接收数据的回调函数叫做onReqPriceData（详见4.22）。Api提供行情的方式是调用该接口主动获取。

当前通过API提供的是L1行情，如果需要使用L2数据，请和我们联系。